Schwab

Programmierung und Bedienung des digitalen Kleinrechenautomaten C 8205

Heft 1
Grundlagen der Programmierung

Teil 2

CELLATRON

3. Rechenoperationen im Dualsystem

Alle mathematischen Berechnungen müssen zur numerischen Auswertung auf die vier rationalen Grundrechenarten - Addition, Subtraktion, Multiplikation und Division - zurückgeführt werden. Für die Rechentechnik gilt es, alle Berechnungen auf die Addition, insbesondere die duale Addition, und einige logische und organisatorische Operationen, zu reduzieren.

3.1. Duale Addition

Die Addition zweier Ziffern (also zweier einstelliger Zahlen) kann durch sogenannte Halbadder (Abb. 9) schematisch dargestellt werden. Man erhält eine Spaltensumme s und einen Übertrag ü. Die dabei entstehenden Werte lassen sich unmittelbar ausrechnen und man erhält für das Dezimalsystem die beiden nachfolgend angeführten Ergebnismatrizen.

Übertragungsmatrix

ü	0	1	2	3	4	5	6	7	8	9
0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	1
2	0	0	0	0	0	0	0	0	1	1
3	0	0	0	0	0	0	0	1	1	1
4	0	0	0	0	0	0	1	1	1	1
5	0	0	0	0	0	1	1	1	1	1
6	0	0	0	0	1	1	1	1	1	1
7	0	0	0	1	1	1	1	1	1	1
8	0	0	1	1	1	1	1	1	1	1
9	0	1	1	1	1	1	1	1	1	1

Spaltensummenmatrix

s	0	1	2	3	4	5	6	7	8	9
0	0	1	2	3	4	5	6	7	8	9
1	1	2	3	4	5	6	7	8	9	0
2	2	3	4	5	6	7	8	9	0	1
3	3	4	5	6	7	8	9	0	1	2
4	4	5	6	7	8	9	0	1	2	3
5	5	6	7	8	9	0	1	2	3	4
6	6	7	8	9	0	1	2	3	4	5
7	7	8	9	0	1	2	3	4	5	6
8	8	9	0	1	2	3	4	5	6	7
9	9	0	1	2	3	4	5	6	7	8

Ähnlich lassen sich Matrizen für andere Zahlensysteme herstellen, was dem Leser zur Übung empfohlen wird.

Besonders einfach wird die Situation im Dualsystem. Hier gibt es für die Addition der Ziffern nur vier Möglichkeiten:

0 + 0 = (0) 0
0 + L = (0) L
L + 0 = (0) L
L + L = (L) 0

und somit die Matrizen

ü	0	L
0	0	0
L	0	L

s	0	L
0	0	L
L	L	0

Komplizierter ist die Situation, wenn mehrstellige Zahlen addiert werden. Zwar ist auch dabei die Addition der Ziffer die Grundlage, doch entsteht die Schwierigkeit dadurch, daß hier der Übertrag mit berücksichtigt werden muß, der ggf. mehrfach auftreten kann. Beispielsweise erhält man das Ergebnis der Addition von 576 und 729 erst nach drei Schritten, wie folgende Rechnung zeigt:

```
    5 7 6
+   7 2 9
-------------
    2 9 5     Spaltensumme
+ 1 0 1       Übertrag
-------------
  1 2 0 5     Spaltensumme
+ 0 0 1 0     Übertrag
-------------
  0 1 3 0 5   Spaltensumme
+ 0 0 0 0
-------------
  1 3 0 5     Ergebnis
```

Erst wenn in keiner Spalte ein Übertrag entsteht, ist die Addition abgeschlossen.
Hieraus folgt:

I. Die Addition mehrstelliger Zahlen erfolgt für jeden Stellenwert durch Addition der Ziffernwerte jeder Stelle.

II. Werden zwei Ziffern addiert, entsteht ein zwei-

stelliges Ergebnis, bestehend aus der Spaltensumme s und dem Übertrag ü, wobei der Übertrag auch den Wert Null annehmen kann.

III. Diesen Prozeß kann man durch den Halbadder technisch realisieren.
Die Werte für die Spaltensumme und für den Übertrag kann einer Spaltensummenmatrix und einer Übertragungsmatrix entnommen werden.

IV. Für die volle Addition müssen aber drei Eingänge Berücksichtigung finden:

a_k Ziffer a der k-ten Spalte
b_k Ziffer b der k-ten Spalte
$ü_{k-1}$ Übertrag aus der (k - 1)-ten Spalte

Als Ergebnis erhält man hier ebenfalls

a_k die Spaltensumme der k-ten Spalte
$ü_k$ den Übertrag der k-ten Spalte, der in die (k - 1)-te Spalte geleitet werden muß.

Zur Ausführung dieser Operation benötigt man für alle Stellenwerte, außer der letzten, den auf Abb. 10 skizzierten Adder. Mittels Halbaddern und Addern läßt sich dann ein Addierwerk aufbauen, wie es für 6-stellige Zahlen und Parallelverarbeitung die Abb. 11 zeigt.

Besonders einfach wird die Situation wiederum beim Rechnen im Dualsystem.

Hier wird der Halbadder durch folgende Wertetabelle charakterisiert:

a	b	ü	s
0	0	0	0
0	L	0	L
L	0	0	L
L	L	L	0

Für den Adder ergibt sich

a_k	b_k	$\ddot{u}_{k-1}$	$\ddot{u}_k$	s_k
0	0	0	0	0
0	L	0	0	L
L	0	0	0	L
L	L	0	L	0
0	0	L	0	L
0	L	L	L	0
L	0	L	L	0
L	L	L	L	L

Dann kann der Adder aber durch die auf Abb. 12 skizzierte Verbindung aufgebaut werden. Dies zeigt folgende Tabelle:

1	2	3	4	5	6	7	8	9
a_k	b_k	s_{kp}	$\ddot{u}_{k-1}$	$\ddot{u}_{kp}$	$\ddot{u}_{ks}$	n	$\ddot{u}_k$	s_k
0	0	0	0	0	0	0	0	0
0	L	L	0	0	0	0	0	L
L	0	L	0	0	0	0	0	L
L	L	0	0	L	0	0	L	0
0	0	0	L	0	0	0	0	L
0	L	L	L	0	L	0	L	0
L	0	L	L	0	L	0	L	0
L	L	0	L	L	0	0	L	L

1 und 2 bildet den Eingang des HA 1. Dazu kommt als Eingang der Wert der Spalte 4, der gleich in den HA 2 gegeben wird. In den HA 2 geht die primäre Spaltensumme des HA 1. HA 2 liefert die Gesamtspaltensumme der Spalte 9 und den Sekundärübertrag der Spalte 6. Der Primärübertrag des HA 1, Spalte 5 und Spalte 6, gehen in den HA 3. Dieser liefert als Spaltensumme den Gesamtübertrag Spalte 8 und der Übertrag ist hier stets 0, wie Spalte 7 ausweist. Daher wird HA 3 später durch ein einfacheres Bauelement ersetzt. Die Spalten 1, 2 und 4 als Eingabe und 8, 9 als Ausgabe zeigen die Übereinstimmung mit der darüberstehenden Tabelle für den Adder.

Gelingt es also, eine Schaltung für den Halbadder aufzustellen, kann daraus der Adder und das Addierwerk für Parallelverarbeitung aufgebaut werden. Die Schaltung des Halbadders wird später erarbeitet.

Nunmehr gilt es zu zeigen, daß die drei anderen Rechenoperationen - Subtraktion, Multiplikation und Division - auf die Addition zurückgeführt - also mittels eines Rechenwerkes und somit mittels Halbadder ausgeführt - werden können.

3.2. Duale Subtraktion

Ähnlich wie bei der Addition könnte die Subtraktion diskutiert werden. Es würde ebenfalls gelingen, dafür allgemeingültige Gesetzmäßigkeiten und Regeln, also einen Algorithmus, aufzustellen. Die Subtraktion soll jedoch unter Anwendung der Addition bewältigt werden.

Das ist relativ einfach durch folgende Überlegungen möglich. Man bildet, wenn a und b zwei beliebige positive Zahlen sind,

$$a - b = a + (0 - b).$$

In diesem Ausdruck wird über die 0 wie folgt verfügt:

$$0 = 10^n - 10^n ,$$

wobei n so gewählt werden muß, daß 10^n um eine Stelle größer ist als die größte der an der Berechnung beteiligten Zahlen. Bei fester Wortlänge wird n gleich der Anzahl der Ziffern gewählt. Damit ergibt sich

$$\begin{aligned} a - b &= a + (0 - b) \\ &= a + (10^n - 10^n - b) \\ &= a + (10^n - b) - 10^n \end{aligned}$$

Beispiel für 6-stellige Dezimalzahlen:

```
      2 3 7  2 4 6      n = 6
-     1 1 3  2 4 3      10^n = 1 000 000
-------------------
      2 3 7  2 4 6
    [ 1 0 0  0 0 0 ]
+   [             ]     Zehnerkomplement
    [-1 1 3  2 4 3 ]
- 1  0 0 0  0 0 0
-------------------
      2 3 7  2 4 6
+     8 8 6  7 5 7      Zehnerkomplement
- 1   0 0 0  0 0 0
-------------------
    1 1 2 4  0 0 3
- 1  0 0 0  0 0 0
-------------------
      1 2 4  0 0 3
```

Formal könnte man (0 - b) bilden, indem man b von Null subtrahiert.

Diesen Ausdruck (0 - b) soll B-Komplement [1])oder, wenn man sich vorerst auf das Dezimalsystem beschränkt, Zehnerkomplement genannt werden.

Beispielsweise ergab sich in der oben durchgeführten Rechnung das Zehnerkomplement für 113243 zu

```
  1 0 0 0  0 0 0
-     1 1 3  2 4 3
-------------------
      8 8 6  7 5 7
```

und damit die Subtraktion:

```
      2 3 7  2 4 6      Subtrahend
-     1 1 3  2 4 3      Minuend
-------------------
      2 3 7  2 4 6      Subtrahend
+     8 8 6  7 5 7      Zehnerkomplement des Minuenden
-------------------
    1 1 2 4  0 0 3
```

---

[1]) Exakt wird B^n - b als B-Komplement bezeichnet.

was offensichtlich richtig ist, wenn man die "übergelaufene" 1 nicht weiter berücksichtigt. Sie darf auch nicht beachtet werden, denn nach der umseitig angeführten Rechnung wird sie ja durch die Subtraktion mit 10^n (speziell 10^6) zu Null.

Scheinbar ist vorerst nichts gewonnen. Die Subtraktion scheint nur an eine andere Stelle verlegt worden zu sein. Dem ist nicht so, denn die Bildung des Zehnerkomplements kann nach einer sehr einfachen Vorschrift erfolgen:

I. Suche für die letzte Stelle die Ziffer, die mit den angegebenen die Summe 10 bildet.

II. Suche für alle anderen Stellen die Ziffern, die mit den angegebenen die Summe 9 bilden.

Damit wird die Subtraktion durch folgenden Algorithmus auf eine Addition zurückgeführt:

1. Bilde für die negative Zahl das Zehnerkomplement.
2. Addiere beide Zahlen.
3. Lasse die "überlaufende" 1 unberücksichtigt.

Unbefriedigend ist es jedoch, daß für die Bildung des Komplements zwei Regeln berücksichtigt werden müssen. Das kann vermieden werden, indem man das Neunerkomplement nimmt.

Zur Ermittlung des Neunerkomplements wird nur die eine Vorschrift herangezogen:

Bestimme für alle Stellen die Ziffer, die, zur gegebenen addiert, 9 ergibt:

$$z_k + z'_k = 9.$$

Mit diesem Neunerkomplement kann man ebenfalls subtrahieren, wie folgendes Beispiel andeutet:

```
      2 3 7  2 4 6
-     1 1 3  2 4 3
-------------------
      2 3 7  2 4 6
+     8 8 6  7 5 6
-------------------
    1 1 2 4  0 0 2
+                 1
-------------------
      1 2 4  0 0 3
```

Jetzt muß allerdings der auftretende Überlauf zur letzten Stelle addiert werden.

Dies ergibt sich wiederum aus der allgemeinen Ableitung recht einfach. Man verfügt jetzt in dem Ausdruck (0 - b) über die Null folgendermaßen:

$$0 = (10^n - 1) - (10^n - 1)$$

Das liefert uns dann:

$$\begin{aligned} a - b &= a + (0 - b) \\ &= a + (10^n - 1) - (10^n - 1) - b \\ &= a + (10^n - 1) - b - (10^n - 1) \\ &= a + (10^n - 1) - b - 10^n + 1 \end{aligned}$$

$(10^n - 1) - b$ ist jetzt das Neunerkomplement, denn von einer Zahl, die aus genau n Neunerziffern besteht, wird b subtrahiert.

Beispiel für 6-stellige Dezimalzahlen:

```
      2 3 7  2 4 6      n = 6
-     1 1 3  2 4 3      10^6 - 1 = 999 999
-------------------
      2 3 7  2 4 6
    [ 1 0 0  0 0 0 ]
+   [ -         1 ]     Neunerkomplement
    [ - 1 1 3  2 4 3 ]
- 1   0 0 0  0 0 0
+                 1
-------------------
      2 3 7  2 4 6
    [ + 9 9  9 9 9 ]    Neunerkomplement
+   [ - 1 1  3 2 4 3 ]
- 1   0 0 0  0 0 0
+                 1
-------------------
```

```
      2 3 7  2 4 6
+     8 8 6  7 5 6     Neunerkomplement
- 1 0 0 0  0 0 0 0
+                 1
-------------------
      1 2 4  0 0 2
+                 1
-------------------
      1 2 4  0 0 3
```

Ein Berechnungsalgorithmus für die Addition und Subtraktion ließe sich nun wie folgt formulieren:

I. Überprüfe das Vorzeichen jeder beteiligten Zahl. Ist es +, folgt II. Ist es -, folgt III.

II. Lasse die Zahl unverändert, darauf folgt IV.

III. Bilde das Neunerkomplement, dann folgt IV.

IV, Addiere die Zahlen!

V. Addiere den eventuell auftretenden Überlauf zur letzten Stelle. Dann folgt STOPP.

In gleicher Weise könnte man im Oktalsystem verfahren. Allerdings entspricht dort dem Zehnerkomplement ein Achterkomplement und dem Neunerkomplement ein Siebenerkomplement. Ersteres nennt man auch B-Komplement, wobei B die Basis des Zahlensystems ist und letztere (B - 1)-Komplement.

Im Dualsystem verwendet man das Zweier- und das Einerkomplement.

Besonders einfach ist die Arbeitsweise mit dem Einerkomplement.

Die Vorschrift für das Bilden des Einerkomplements lautet:

Bestimme für alle Stellen die Dualziffer, die zur ge-

gebenen addiert ein L ergibt

$$d_k + d_{k'} = L.$$

Es sollen alle möglichen Fälle durchprobiert werden. Ist $d_k = L$, so ergibt sich

$$L + d_{k'} = L.$$

Folglich muß $d_{k'} = 0$ sein.

Ist $d_k = 0$, folgt

$$0 + d_{k'} = L,$$

also muß $d_{k'} = L$ sein.

Das heißt aber doch, daß immer dort, wo in der Originalzahl ein L steht, im Komplement eine 0 und für die 0 ein L zu setzen ist. Diese Vorschrift

$$0 \longrightarrow L \quad ^{1)}$$
$$L \longrightarrow 0$$

nennt man Negation. Die Negation wird durch Überstreichen gekennzeichnet, also

$$\overline{0} = L \quad ^{2)}$$
$$\overline{L} = 0$$

Damit kommt man im Dualsystem zu der einfachen Regel für das Bilden des Einerkomplements:

Das Einerkomplement wird im Dualsystem durch Negation jeder Ziffer gebildet, denn dann gilt:

$$d_k + d_{k'} = L$$

Die Subtraktion kann dann in der folgenden Form ausgeführt werden:

---

1) Das Zeichen $\longrightarrow$ soll hier "wird zu" bedeuten

	1	3	+	0	L	L	0	L
-		6	-	0	0	L	L	0
	1	3	+	0	L	L	0	L
+	9	3	+	L	L	0	0	L
1	0	6	L	0	0	L	L	0
		1	+					L
		7	+	0	0	L	L	L

Schließt man das Vorzeichen nach der Vorschrift

+ ≙ 0
- ≙ L

in die Rechnung ein, so kann ein Algorithmus für Addition und Subtraktion aufgestellt werden, wie er auf Abb. 13 dargestellt ist. Dieser Algorithmus soll an Beispielen erprobt werden.

Zur besseren Übersicht werde angenommen, daß Worte fester Länge zu 6 Bits mit einem Bit für das Vorzeichen benutzt werden. Zudem werde angenommen, daß es sich um ganze Zahlen handelt (Komma also hinter der letzten geltenden Ziffer).

Dann ist:

	i	Z i	sign P	sign P/2	sign P	S'	Ü	Ü(1)	S	sign (2)	Z dual	Z dez.
1. Fall	0											
+13+6=+19	1	+0LLOL	P̄	0	P							
	2	+00LLO	P̄	0	P̄	0 L00LL	0	00000	0 L00LL	0	+L00LL	+19
2. Fall	0											
+13-6=+7	1	+0LLOL	P̄	0	P							
	2	-00LLO	P	L								
		+LL00L			P̄	0 00LLO	L	0000L	0 00LLL	0	+00LLL	+ 7
3. Fall	0											
-13+6=-7	1	-0LL0L	P	L								
		+L00L0			P							
	2	+00LLO	P̄	0	P̄	L LL000	0	00000	L LL000	L	-00LLL	- 7
4. Fall	0											
-13-6=-19	1	-0LL0L	P	L								
		+L00L0			P							
	2	-00LLO	P	L								
		+LL00L			P̄	L 0L0LL	L	0000L	L 0LL00	L	-L00LL	-19
5. Fall	0											
-13+13=0	1	-0LL0L	P	L								
		+L00L0			P							
	2	+0LL0L	P̄	0	P̄	L LLLLL	0	00000	L LLLLL	0	-00000	- 0

In ähnlicher Weise kann auch ein Algorithmus mit dem Zweierkomplement aufgebaut werden. Dabei ist die Bildung des Komplements dadurch schwieriger, weil die Negation für die Einerstelle der Zahl (Ganzzahligkeit angenommen) nicht ausgeführt werden darf. Dafür entfällt das Addieren des Überlaufes zur letzten Stelle.

Prinzipiell ist es aber gelungen, die Subtraktion auf die Addition und die Komplementbildung zurückzuführen. Betrachtet man dies insbesondere im Dualsystem, so vereinfacht sich das Zurückführen der Subtraktion auf die Addition und Negation.

1.3. Duale Multiplikation

Es bereitet keine Schwierigkeiten, die Multiplikation auf die Addition zurückzuführen.

Das zeigt ein einfaches Beispiel:

```
      2 1 3 4 7 · 2 3 1
      ---------------
      2 1 3 4 7
+   2 1 3 4 7
+   2 1 3 4 7
+   2 1 3 4 7
+ 2 1 3 4 7
+ 2 1 3 4 7
-----------------
  4 9 3 1 1 5 7
```

Man benötigt also die Addition und Linksverschiebung. Besonders einfach wird dies, wenn man im Dualsystem rechnet. Da nur die Ziffern 0 oder L auftreten können, wird bei jeder Linksverschiebung entweder bei L der zweite Faktor addiert, oder es wird bei 0 gleich anschließend die nächste Linksverschiebung ausgeführt.

Beispiel: 5 · 9 = 45

```
          0 0 L 0 L · 0 L 0 0 L
          -----------------------
          0 0 L 0 L
        0 0 0 0 0
      0 0 0 0 0
    0 0 L 0 L
  0 0 0 0 0
  -------------------
        L 0 L L 0 L
```

Damit ist es gelungen, die Multiplikation auf die Addition und Linksverschiebung zurückzuführen.

3.4. Duale Division

Auch die Division läßt sich, allerdings erst über die Subtraktion, auf die Addition zurückführen.

Beispiel:

```
    3771120 : 31426 = 120
-   31426
    -----
     62852
-    31426
    -----
     31426
-    31426
    -----
        00
```

Der Divisor wird vom höchstmöglichen Stellenwert des Dividenden so oft subtrahiert, bis der Rest kleiner als der Divisor ist. Dann wird um eine Stelle nach rechts verschoben und das Verfahren bis zur gewünschten Genauigkeit wiederholt. Damit wird die Division zurückgeführt auf die Subtraktion und Rechtsverschiebung und so auf die Addition, Komplementbildung und Rechtsverschiebung.

Nunmehr ist es gelungen, die rationalen Rechenoperationen auf

- Addition
- Komplementbildung und
- Stellenverschiebung (d.h. Multiplikation mit B respektive B^{-1})

zu reduzieren.

Betrachtet man nur das Dualsystem, so vereinfacht sich dies auf duale Addition, Negation und Stellenverschiebung (hier also Multiplikation mit 2 oder 2^{-1}).

3.5. Rechnen mit dual codierten Dezimalzahlen

Eine Besonderheit ergibt sich beim Rechnen mit dual codierten Dezimalzahlen.

Werden zwei dual codierte Dezimalzahlen addiert, dabei soll hier nur die direkte dezimal duale Codierung betrachtet werden - dann entstehen Schwierigkeiten bei der Bildung des Übertrages. Ist der Übertrag 0, können die Dualtetraden unmittelbar addiert werden.

3	0011	1	0001	3	0011
+ 6	+ 0110	+ 2	+ 0010	+ 5	+ 0101
9	1001	3	0011	8	1000

Tritt ein Übertrag auf, so wird das Ergebnis verfälscht. Es entstehen Dualtetraden, die bei vorgegebener Codierung keiner Dezimalziffer entsprechen, beispielsweise:

3	0011	6	0110	9	1001
+ 7	+ 0111	+ 5	+ 0101	+ 6	+ 0110
10	1010	11	1011	15	1111

wobei offensichtlich kein Übertrag erzeugt wird. Andererseits ergibt sich zwar ein Übertrag, aber die entstandene Spaltensumme ist falsch.

Zum Beispiel:

7	0111	9	1001	8	1000
+ 9	+ 1001	+ 9	+ 1001	+ 9	+ 1001
16	(1)0000	18	(1)0010	17	(1)0001

Die letzten Beispiele zeigen, daß die Verfälschung gerade sechs Einheiten beträgt.

Diese Erscheinung ergibt sich aus der Tatsache, daß im Dezimalsystem bei der Spaltensumme 10 ein Übertrag erfolgt. Die Darstellung der Zahl in Dualtetraden entspricht aber einem Hexadezimalsystem, und der Übertrag erfolgt, wenn 15 überschritten wird.

Zur Überwindung dieser Schwierigkeit ist eine soge-

nannte tetradische Korrektur auszuführen.

Übersteigt die Spaltensumme bei der Addition dezimal-dual codierter Zahlen den Wert 9, so ist zur Tetrade der Spaltensumme die Korrekturtetrade

6 = OLLO

zu addieren.

Damit ergibt sich dann auch der richtige Übertrag. Korrigiert man die vorhin betrachteten Beispiele, so folgt:

ØØLL	OLLO	LOOL
+ OLLL	+ OLOL	+ OLLO
LOLO	LOLL	LLLL
+ OLLO	+ OLLO	+ OLLO
(L)0000	(L)000L	(L)OLOL

OLLL	LOOL	LOOO
+ LOOL	+ LOOL	+ LOOL
(L)0000	(L)OOLO	(L)OOOL
+ OLLO	+ OLLO	+ OLLO
(L)OLLO	(L)LOOO	(L)OLLL

was offensichtlich zu richtigen Ergebnissen führt. Ähnliche Korrekturen müssen auch für die Aiken- und die Dreiexzeß-Verschlüsselung vorgenommen werden.

Für die Addition der Gesamtzahl empfiehlt es sich, spaltenweise die korrigierte Summe zu bilden. Der dabei entstehende Überlauf kann dann unmittelbar in der nächsthöheren Spalte berücksichtigt werden.

Beispiel:

	2	3	7	2 4
+	3	2	6	9 1
	5	6	4	1 5

3LO	OOLL	OLLL	OOLO	OLOO	1. Summand
OOLL	OOLO	OLLO	LOOL	OOOL	2. Summand
OLOL	OLOL	LLOL	LOLL	OLOL	unkorrigierte Summe
		OLLO	OLLO		Korrekturfaktor
	OOOL	OOOL			
OLOL	OLLO	OLOO	OOOL	OLOL	korrigierte Summe

Die Subtraktion kann auch hier auf die Komplementbildung und Addition zurückgeführt werden. Das bereitet bei der direkten Codierung jedoch einige Schwierigkeiten, weil das Komplement berechnet werden muß.

Die Aiken- und Dreierexzeß-Codierung können die Komplemente durch Negation bilden, haben dafür aber eine kompliziertere tetradische Korrektur.

Multiplikation und Division sind sinngemäß auszuführen.

Die Boolesche Algebra

Die bisherigen Betrachtungen zeigten, daß die gesamte Rechenarbeit auf die rationale Rechenoperation Addition sowie Komplementbildung und Verschiebung zurückgeführt werden kann. Die Addition ließ sich über Halbadder ermöglichen. Es verbleibt somit zu zeigen, wie sich Halbadder aus solchen mathematischen Funktionen aufbauen lassen, die auch technisch ausgeführt werden können.

Hierfür gab der englische Mathematiker George Boole die Gesetzmäßigkeiten in der nach ihm benannten Booleschen Algebra an.

Er geht von der bereits erwähnten zweiwertigen Logik aus. Jede logische Variable kann also nur die Werte "wahr" oder "falsch" annehmen. Dann untersuchte er Verknüpfungen zweier Variabler, von denen 16 definiert werden können. Die Boolesche Algebra läßt sich auf

drei Operationen aufbauen,

- die Konjunktion oder "UND"-Verknüpfung
- die Disjunktion oder "ODER"-Verknüpfung
- die Negation,

Bei der Konjunktion werden zwei logistische Variable durch die Verknüpfung "UND" verbunden. Welchen Wahrheitswert muß dann die verknüpfte Aussage erhalten?

Das soll am Beispiel abgeleitet werden.
Es wird vereinbart:

"(A) wenn es Sonntag ist UND
(B) wenn die Sonne scheint, trifft sich das Kollektiv zum Ausflug."

Hier sind die Aussagen (A) und (B) konjunktiv, also durch UND verknüpft.

Das Ergebnis der Verknüpfung ermittelt man durch Probieren.

I. A sowie B seien "wahr". Es ist Sonntag, und die Sonne scheint, dann wird, wie vereinbart, gefahren.

II. A sei "unwahr", es ist also kein Sonntag, B sei "wahr", die Sonne scheint, dann wird nicht gefahren.

III. A sei "wahr", B "unwahr", jetzt ist zwar Sonntag, aber die Sonne scheint nicht, dann wird auch nicht gefahren.

IV. A sei "unwahr" und B sei "unwahr", jetzt ist kein Sonntag, und die Sonne scheint auch nicht, dann wird selbstverständlicherweise ebenfalls nicht gefahren.

Setzt man das Fahren als "wahr", das Nicht-Fahren-Sollen als "unwahr", so erhält man mit Einführung des Zeichens $\wedge$ als "UND" folgende Relationen:

A "wahr" , B "wahr" => A ∧ B "wahr"
A "unwahr", B "wahr" => A ∧ B "unwahr"
A "wahr" , B "unwahr" => A ∧ B "unwahr"
A "unwahr", B "unwahr" => A ∧ B "unwahr".

Setzt man

"wahr" = L und
"unwahr" = 0

läßt sich eine einfache Schreibweise mit Hilfe dualer Ziffern einführen gemäß:

L ∧ L = L
0 ∧ L = 0
L ∧ 0 = 0
0 ∧ 0 = 0

(lies: L und L ergibt L)
oder unter Verwendung einer Ergebnismatrix:

∧	0	L
0	0	0
L	0	L

Ergebnismatrix der Konjunktion

Die Disjunktion ist eine logische Funktion, bei der zwei Variable durch ein nicht ausschließendes ODER verknüpft werden, also nicht im Sinne des lateinischen Sprichwortes "aut amicus, aut inamicus" - "entweder Freund oder Feind". Auch hier die Ableitung als Beispiel.

Als Zeuge eines Verkehrsunfalls gibt man die nicht erkannte Farbe des unfallverursachenden Wagens wie folgt an:

"Die Farbe des Wagens war grün ODER blau".

Nach Feststellen des Wagens könnten sich folgende Fälle ergeben, mit ∨ als Zeichen der Disjunktion:

I. Der Wagen ist rot, also
A "unwahr"; B "unwahr" = $A \vee B$ "unwahr"

II. Der Wagen ist grün:
A "wahr"; B "unwahr" = $A \vee B$ "wahr"

III. Der Wagen ist blau:
A "unwahr"; B "wahr" = $A \vee B$ "wahr"

IV. Der Wagen ist grün;
unten blau:
A "wahr"; B "wahr" = $A \vee B$ "wahr"

Führt man auch hier als Symbol Dualzahlen ein, ergibt sich

$$0 \vee 0 = 0$$
$$L \vee 0 = L$$
$$0 \vee L = L$$
$$L \vee L = L$$

oder als Ergebnismatrix der Disjunktion:

$\vee$	0	L
0	0	L
L	L	L

Zu den eben ermittelten Booleschen Funktionen benötigt man noch die bereits bekannte Negation nach der Festlegung

$$\bar{L} = 0$$
$$\bar{0} = L$$

Die besprochenen Funktionen der Booleschen Algebra lassen sich sehr einfach technisch realisieren.

Man betrachtet dazu als Aussage lediglich das Vorhandensein eines Stromimpulses. Diesem Fall soll der Wahrheitswert L zugeschrieben werden. Das Ausbleiben eines Stromimpulses soll dem Wahrheitswert 0 entsprechen. Als Schaltelement seien vorerst einfache Schalter

gedacht, die zwei Stellungen einnehmen können:

Arbeitsstellung — der Impuls wird weitergeleitet (L)
oder
Ruhestellung — der Impuls wird nicht weitergeleitet (0).

Dann liefert die auf Abb. 14 dargestellte Schaltung die Konjunktion, denn es gilt:

1. Schalter 1 : 0
   Schalter 2 : 0
   dann wird der Impuls nicht weitergeleitet, also (0)

2. Schalter 1 : L
   Schalter 2 : 0
   dann wird der Impuls nicht weitergeleitet, also (0)

3. Schalter 1 : 0
   Schalter 2 : L
   dann wird der Impuls auch nicht weitergeleitet, also (0)

4. Schalter 1 : L
   Schalter 2 : L
   dann wird der Impuls weitergeleitet, da der Stromfluß geschlossen ist, also (L)

Das liefert als Ergebnismatrix die Konjunktion

A	0	L
0	0	0
L	0	L

die man somit durch Serienschaltung der Schaltelemente erhalten würde.

Die Disjunktion erhält man durch Parallelschalten zweier Schaltelemente (Abb. 15).

Es ergeben sich die Fälle:

1. Schalter 1 : 0
   Schalter 2 : 0

Der Impuls wird nicht weitergeleitet, da beide Zweige unterbrochen sind, also (O)

2. Schalter 1 : L
   Schalter 2 : O
   Der Impuls wird weitergeleitet, wobei er über den durch Schalter 1 geschlossenen Zweig fließen kann, also (L)

3. Schalter 1 : O
   Schalter 2 : L
   Der Impuls wird weitergeleitet, wobei er über den durch Schalter 2 geschlossenen Zweig fließen kann, also (L)

4. Schalter 1 : L
   Schalter 2 : L
   Der Impuls wird weitergeleitet. Beide Zweige sind geschlossen, also (L)

Faßt man das Ergebnis in einer Matrix zusammen, ergibt sich die Disjunktion gemäß:

V	O	L
O	O	L
L	L	L

Für die Negation ist die Schaltung noch einfacher. Hier braucht man lediglich die Leitung am Ruhekontakt anzuschließen, dann wird das Ergebnis negiert, denn bei Arbeitsstellung wird dann kein Impuls weitergeleitet, also O, während bei Ruhestellung ein Impuls weitergeleitet wird, also L.
Abb. 16 zeigt das Modell.

Aus diesen Grundschaltungen, die für die weiteren Betrachtungen in der auf Abb. 17 dargestellten Form verwendet werden, läßt sich nun ein Halbadder aufbauen.

Dazu muß es gelingen, die Matrizen

ü	0	L
0	0	0
L	0	L

s	0	L
0	0	L
L	L	0

der dualen Addition durch Funktionen der Booleschen Algebra auszudrücken.

Für den Übertrag ist das einfach.
Er entspricht der Konjunktion:

ü	0	L
0	0	0
L	0	L

$\wedge$	0	L
0	0	0
L	0	L

und ist durch eine Serienschaltung von zwei Schaltelementen darstellbar.

Schwieriger ist dies für die Spaltensummen. Weder die Disjunktion noch die Konjunktion entsprechen der Ergebnismatrix.

Da gilt:

$$a \wedge \bar{a} = 0$$
$$a \vee \bar{a} = L$$

ist

$\overline{a \wedge b}$	0	L
0	L	L
L	L	0

bildet man

$$(a \vee b) \wedge (\overline{a \wedge b}) = z,$$

so ergibt sich eine mögliche Realisierung der Spaltensummenmatrix, denn es ist für

I. $a = 0;\ b = 0$

$$\begin{aligned} z &= (a \vee b) \wedge \overline{(a \wedge b)} \\ &= (0 \vee 0) \wedge \overline{(0 \wedge 0)} \\ &= 0 \wedge \overline{0} \\ &= 0 \wedge L \\ &= 0 \end{aligned}$$

II. $a = L;\ b = 0$

$$\begin{aligned} z &= (a \vee b) \wedge \overline{(a \wedge b)} \\ &= (L \vee 0) \wedge \overline{(L \wedge 0)} \\ &= L \wedge \overline{0} \\ &= L \wedge L \\ &= L \end{aligned}$$

III. $a = 0;\ b = L$

$$\begin{aligned} z &= (a \vee b) \wedge \overline{(a \wedge b)} \\ &= (0 \vee L) \wedge \overline{(L \wedge 0)} \\ &= L \wedge \overline{0} \\ &= L \wedge L \\ &= L \end{aligned}$$

IV. $a = L;\ b = L$

$$\begin{aligned} z &= (a \vee b) \wedge \overline{(a \wedge b)} \\ &= (L \vee L) \wedge \overline{(L \wedge L)} \\ &= L \wedge \overline{L} \\ &= L \wedge 0 \\ &= 0 \end{aligned}$$

Somit

z	0	L
0	0	L
L	L	0

Wohlbemerkt, das ist eine Möglichkeit, um eine Spaltensumme technisch zu realisieren.

Dann läßt sich der Halbadder wie folgt realisieren:

$$\begin{aligned} \ddot{u} &:= a \wedge b \\ s &:= (a \vee b) \wedge \overline{(a \wedge b)} \end{aligned}$$

das ist für den Übertrag offensichtlich, und für die Spaltensumme wurde das Ergebnis abgeleitet.
Dem entspricht Abb. 18.

Da gilt

$$\begin{aligned} a \vee \bar{a} &= L \\ a \wedge \bar{a} &= 0 \\ L \wedge a &= a \\ \bar{a} \vee \bar{b} &= \overline{a \wedge b}, \end{aligned}$$

denn es ist

$\bar{a} \vee \bar{b}$	0	L
0	L	L
L	L	0

$\overline{a \wedge b}$	0	L
0	L	L
L	L	0

folgt

$$\begin{aligned} (a \vee b) \wedge (\overline{a \wedge b}) &= (a \vee b) \wedge (\bar{a} \vee \bar{b}) \\ &= (a \wedge \bar{a}) \vee (a \wedge \bar{b}) \vee \\ &\quad (\bar{a} \wedge \bar{b}) \vee (b \wedge \bar{b}) \\ &= (a \wedge \bar{b}) \vee (\bar{a} \wedge b) \end{aligned}$$

Das liefert eine weitere Realisierungsmöglichkeit für den Halbadder über die Formeln

$$\begin{aligned} \ddot{u} &:= (a \wedge b) \\ s &:= (a \wedge \bar{b}) \vee (\bar{a} \wedge b) \end{aligned}$$

Das ist für ü ebenfalls offensichtlich, und s ergibt sich nach folgenden Überlegungen:

$$\begin{aligned} a = 0;\ b &= 0 \\ (a \wedge \bar{b}) \vee (\bar{a} \wedge b) &= (0 \wedge \bar{0}) \vee (\bar{0} \wedge 0) \\ &= (0 \wedge L) \vee (L \wedge 0) \\ &= 0 \vee 0 \\ &= 0 \end{aligned}$$

$a = L;\ b = 0$

$$(a \wedge \bar{b}) \vee (\bar{a} \wedge b)$$
$$= (L \wedge \bar{0}) \vee (\bar{L} \wedge 0)$$
$$= (L \wedge L) \vee (0 \wedge 0)$$
$$= L \vee 0$$
$$= L$$

$a = 0;\ b = L$

$$(a \wedge \bar{b}) \vee (\bar{a} \wedge b)$$
$$= (0 \wedge \bar{L}) \vee (\bar{0} \wedge L)$$
$$= (0 \wedge 0) \vee (L \wedge L)$$
$$= 0 \vee L$$
$$= L$$

$a = L;\ b = L$

$$(a \wedge \bar{b}) \vee (\bar{a} \wedge b)$$
$$= (L \wedge \bar{L}) \vee (\bar{L} \wedge L)$$
$$= (L \wedge 0) \vee (0 \wedge L)$$
$$= 0 \vee 0$$
$$= 0$$

Also als Ergebnismatrix

$(a \wedge \bar{b}) \vee (\bar{a} \wedge b)$	0	L
0	0	L
L	L	0

Dem entspricht Abb. 19.

Mit Hilfe der Booleschen Algebra können noch viele Formeln zur Schaltung eines Halbadders gewonnen werden, wobei die entsprechenden Schaltungen für die unterschiedlichen bistabilen Schaltelemente mehr oder weniger vorteilhaft sind.

Für die theoretischen Untersuchungen bei der Entwicklung und Konstruktion von Rechenautomaten gibt es eine eigene wissenschaftliche Disziplin, die Schaltalgebra. Ihre Grundlage bildet die Boolesche Algebra.

Eine Aufgabe der Schaltalgebra ist es auch, die für die jeweils verwendeten Schaltelemente günstigste Schaltung zu ermitteln.

An diese werden verschiedene Anforderungen gestellt, wie hohe Betriebssicherheit, hohe Operationsgeschwindigkeit des geschalteten Systems, minimale Anzahl an Schaltelementen und vieles andere. Diese Forderungen widersprechen sich teilweise. Mit Hilfe der Schaltalgebra kann die Einzelschaltung wie auch die Schaltung des gesamten Automaten aufgestellt, untersucht und optimiert werden.

Insbesondere kann nunmehr der Adder endgültig aufgebaut werden, indem der HA 3 der Abb. 12 durch eine Disjunktion ersetzt wird. Der Adder erhält damit die auf Abb. 20 dargestellte Form, der folgende Tabelle entspricht:

a_i	b_i	$S_{pr\ i}$	$U_{ges\ i-1}$	$U_{pr\ i}$	$U_{sek\ i}$	$U_{ges\ i}$	$S_{ges\ i}$
0	0	0	0	0	0	0	0
L	0	L	0	0	0	0	L
0	L	L	0	0	0	0	L
L	L	0	0	L	0	L	0
0	0	0	L	0	0	0	L
L	0	L	L	0	L	L	0
0	L	L	L	0	L	L	0
L	L	0	L	L	0	L	L

Damit ist die Lücke geschlossen.

Technisch lassen sich die Booleschen Funktionen durch bistabile Schaltelemente aufbauen.

Mittels der Booleschen Funktionen kann ein Halbadder konstruiert werden, aus dem sich dann der Adder und das Addierwerk aufbauen lassen.

Auf Additionen (dualen) und logischen sowie organisatorische Operationen lassen sich wiederum die rationalen Rechenoperationen zurückführen, die Grundlage der Numerischen Mathematik sind.

4. Beschreibung einer programmgesteuerten digitalen Rechenanlage

4.1. Arbeitsweise digitaler Rechenautomaten

Digitale Rechenmaschinen sind Geräte, die Zahlen ziffernweise, d.h. diskret darstellen und verarbeiten, so wie das aus der Algebra bekannt ist. Im Gegensatz dazu werden in der analogen Rechentechnik stetig veränderliche physikalische Größen zur Darstellung der Zahlen benutzt. Das wohl verbreitetste Analogrechengerät ist der Rechenstab.

Als Rechenautomaten werden hier ausschließlich digitale Rechenmaschinen mit programmgesteuertem Arbeitsablauf bezeichnet.

4.1.1. Programmsteuerung

Der entscheidende Schritt bei der Übernahme geistiger Routinearbeiten durch Maschinen war die Einführung der Programmsteuerung. Diese bewirkt, daß ein vom Menschen einmal festgelegter Arbeitsablauf von der Maschine beliebig oft solange ausgeführt werden kann, bis der Mensch der Maschine eine andere Arbeitsweise vorschreibt.

Das Festlegen des Arbeitsablaufes nennt man Programmierung. Vertreter programmgesteuerter Maschinen sind z.B. Werkzeugmaschinen, die durch Lochstreifen gesteuert werden.

Moderne programmgesteuerte Rechenautomaten lesen das Programm von maschinell lesbaren Datenträgern (Lochband, Lochkarte und Magnetband) und speichern es in einem internen Speicher, dem Hauptspeicher des Gerätes. Von dort wird es stückweise abgerufen und abgearbeitet.

4.1.2. Befehle

Der Teil eines Programmes, der gleichzeitig abgerufen wird und dann einen Teilablauf bewirkt, nennt man

einen Befehl.

Jeder Rechenautomat besitzt eine gewisse Menge von Befehlen, die unterschiedliche Wirkungen haben. Diese Menge bezeichnet man als das Befehlssystem des Rechners.

In einem Befehlssystem sind Befehle enthalten, die eine Veränderung von Zahlen bewirken (nicht-organisatorische Befehle), aber auch solche, die nur der Organisierung des Arbeitsablaufes (Veränderung der Reihenfolge der Befehlsabarbeitung, Eingaben, Ausgaben) dienen (Organisationsbefehle).

Ein Befehl kann in Abhängigkeit von seinem Informationsinhalt und dessen Ausnutzung einen sehr verschiedenen Umfang von Funktionsabläufen bewirken.

Um Befehle und Zahlen im Hauptspeicher aufbewahren und bei Bedarf abrufen zu können, unterteilt man ihn in einzelne Zellen, denen eine Nummer, die Adresse, zugeordnet ist. Jede derartige Speicherzelle kann entweder ein einzelnes Zeichen (z.B. eine Dezimalziffer) oder ein ganzes Wort (= Zeichenfolge) aufnehmen.

Jeder Befehl besteht im allgemeinen aus einem Operationsteil und einem Adressenteil. Der Operationsteil bestimmt die Art der auszuführenden Operation, d.h. er legt fest, was zu tun ist (z.B. "addiere", "transportiere"). Der Adressenteil gibt die Speicherzellen an, in denen die an der Operation beteiligten Zahlen zu finden sind.

Nach der Zahl der im Adressenteil eines Befehles enthaltenen Adressen unterscheidet man Ein-, Zwei-, Drei-, Vier- und Fünfadreßmaschinen.

Bei einer Fünfadreßmaschine werden z.B. die Adressen der beiden Operanden, des Ergebnisses, des nächstfolgenden Befehls und eines anderen, bei Vorliegen einer bestimmten Bedingung nächstfolgenden Befehls, angegeben.

Damit werden diese Befehle sehr lang und ihre Ausführung erfordert einen größeren technischen Aufwand.

Bei Einadreßautomaten - z.B. beim C 8205 - besteht der Adressenteil eines Befehls aus nur einer Adresse. Die Verringerung der Adressenzahl wurde durch folgende Maßnahmen erreicht:

1. Abarbeitung der Befehle in einer bestimmten Reihenfolge, so daß die Angabe des nächstfolgenden Befehls entfällt. Diese Reihenfolge der Befehlsabarbeitung kann nur mit Hilfe spezieller Sprungbefehle verlassen werden.

2. Einführung besonderer Sprungbefehle, die in Abhängigkeit von bestimmten Bedingungen den nächstfolgenden Befehl festlegen. Dadurch kann die bedingte Befehlsadresse entfallen.

3. Entnahme eines Operanden (z.B. eines Summanden) aus einem Register (als Register werden Speicherzellen bezeichnet, die nicht nur zur Aufbewahrung, sondern auch zur Verarbeitung von Informationen benutzt werden) und Speicherung des Ergebnisses in diesem Register oder in der durch den anderen Operanden bestimmten Speicherzelle.

   Damit ist nur noch die Angabe der zweiten Operandenadresse notwendig.

Einadreßautomaten haben einen einfacheren technischen Aufbau als Fünfadreßmaschinen, dafür werden aber die Programme etwas länger.

Der Operationsteil eines Befehls kann analytisch oder nicht-analytisch aufgebaut sein.

Bei einem analytischen Befehlscode kommt jeder Dualziffer des Operationsteiles eine bestimmte, von den übrigen unabhängige Bedeutung zu, während beim nicht-analytischen Code jede vollständige Operationszahl eine spezielle Funktion hat.

Die Nachteile des analytischen Befehlscodes sind die große Stellenzahl des Operationsteiles und die zahlreichen nutzlosen Befehlskombinationen. Der nichtanalytische Code dagegen erfordert zur Entschlüsselung einen großen technischen Aufwand. In der Praxis hat sich deshalb weitgehend ein kombiniertes Befehlssystem durchgesetzt, wie man es bei C 8205 findet.

4.1.3. Das Maschinenwort

Wie die bisherigen Ausführungen zeigen, erfolgt in einem digitalen programmgesteuerten Rechenautomaten die Speicherung von Befehlen und Zahlen und die Verarbeitung von Zahlen. Dabei haben sowohl jeder Speicherplatz als auch das (oder die) Register eine begrenzte Stellenkapazität, der bei Maschinen mit fester Wortlänge jeweils ein Wort entspricht. Dieses Wort wird Maschinenwort genannt.

Auf das Maschinenwort beziehen sich alle fest verdrahteten (d.h. durch Schaltungen realisierten) Operationen, z.B. die Addition.

Bei der Speicherung wird in den meisten Automaten nicht zwischen Befehlen und Zahlen unterschieden, so daß Befehle wie Zahlen behandelt und auch verändert werden können.

Das wird insbesondere zur Änderung des Adressenteiles eines Befehls (vgl. 5.4.) ausgenutzt. Mit Hilfe dieser Adressenänderung können Rechenoperationen mit einer Folge von geordnet abgespeicherten Zahlen vorgenommen werden, ohne daß dabei die Befehlsfolge mehrfach mit unterschiedlichen Adressen abgespeichert ist.

4.1.4. Logische Entscheidungen

Eine der wichtigsten Eigenschaften digitaler Rechenautomaten ist ihre Fähigkeit, in Abhängigkeit

von dem Resultat einer vorangegangenen Operation die Reihenfolge der Befehlsabarbeitung verändern zu können. Dabei sind auf die zulässigen Fragestellungen stets nur die Antworten "ja" oder "nein" möglich.

Beispiele für logische Fragestellungen:

Ist das Resultat der letzten Operation negativ?
Überschreitet das Resultat der letzten Rechenoperation den Zahlenbereich des Automaten?
Ist das Ergebnis kleiner als eine Testgröße?
Ist das Resultat gleich einer anderen Zahl?

Wie im Punkt 5.3. gezeigt wird, sind die logischen Entscheidungen ein wichtiges Mittel zur Verkürzung von Programmen und damit zur Verringerung des Speicherplatzbedarfes.

4.2. Struktur programmgesteuerter Rechenautomaten

Bisher wurden die Bedingungen angegeben, die ein Rechenautomat erfüllen muß. Jetzt soll am Beispiel der Einadreßmaschine erläutert werden, welche Struktur der Automat dazu besitzen muß.

Abb. 21 zeigt die Grobstruktur eines Rechenautomaten. Die in der Abbildung eingetragenen Verbindungslinien können auch etwas anders verlaufen. So können z.B. die Ein- und Ausgabegeräte auch mit dem Rechenwerk verbunden sein statt mit dem Speicher.

Der Speicher nimmt die zu verarbeitenden Daten, Zwischenergebnisse, Endresultate usw. und das Programm auf.

Im Rechenwerk werden die arithmetischen und logischen Rechenoperationen ausgeführt. Es enthält meist ein Register, den sogenannten Akkumulator, das zur Speicherung des einen Operanden und des Ergebnisses benutzt wird. Das Rechenwerk kann jedoch auch mehrere Register enthalten.

Die Verbindung des Rechners mit der Außenwelt (dem Menschen) stellen die Ein- und Ausgabegeräte her. Diese Geräte faßt man unter dem Begriff "1. Peripherie" zusammen (periphere Geräte = Randgeräte). Unter der "2. Peripherie" versteht man lochband-, lochkarten- und magnetbanderzeugende Maschinen, die mit dem Rechner nur über den Umweg von Datenträgern (Lochbändern, Lochkarten und Magnetband) in Verbindung stehen.

Das Leitwerk steuert die Zusammenarbeit der verschiedenen Baugruppen. Es analysiert Befehle und veranlaßt, daß die entsprechenden Operationen in der richtigen Reihenfolge ablaufen.

Bei Einadreßmaschinen besteht das Leitwerk im wesentlichen aus einem Befehlsregister und einem Befehlszähler. Im Befehlsregister werden die einzelnen Befehle entschlüsselt und der Befehlszähler steuert die Reihenfolge ihrer Abarbeitung.

In Abb. 22 ist das Schaltbild eines Einadreßautomaten gezeichnet.

Entsprechend dem Stand des Befehlszählers wird ein Befehl aus dem Speicher entnommen und in das Befehlsregister überführt, das aus Operations- und Adressenregister besteht. Bei Rechenbefehlen wird ein in einem Register gespeicherter Operand mit einem aus dem Speicher entnommenen Operanden im Rechenwerk verknüpft. Der zweite Operand wird dabei aus der Speicherzelle entnommen, deren Adresse im Adressenregister angegeben ist. Das Resultat steht danach in einem Register oder in der durch den zweiten Operanden bestimmten Speicherzelle.

Transportbefehle lösen den Transport von Worten zwischen dem Speicher und dem Rechenwerk aus.

Gleichzeitig mit der Durchführung der Operation wird der Inhalt des Befehlszählers um eine Einheit

...c

erhöht, so daß nach Abschluß der Operation der folgende Befehl aus dem Speicher entnommen wird.

Mit Hilfe von Sprungbefehlen kann die Abarbeitung der Befehle in natürlicher Reihenfolge unterbrochen werden. Sie bewirken, daß der Inhalt des Adressenregisters in den Befehlszähler überführt wird.

Bei bedingten Sprungbefehlen ist die Ausführung des Sprunges vom Ergebnis einer vorhergehenden Rechenoperation abhängig. Ist die Sprungbedingung nicht erfüllt, so wird der Inhalt des Befehlszählers um eine Einheit erhöht, so daß die Abarbeitung der Befehle in natürlicher Reihenfolge fortgesetzt wird.

Hinsichtlich der Arbeitsweise programmgesteuerter Rechenautomaten unterscheidet man zwischen Parallel- und Serienarbeitsweise. Dabei bedeutet Serienarbeitsweise, daß die Dualstellen eines Wortes nacheinander verarbeitet werden, während bei der Parallelarbeitsweise die Verarbeitung gleichzeitig erfolgt.

Da bei der Serienarbeitsweise die Bits eines Wortes nacheinander im gleichen Element verarbeitet werden, ist hier der Aufwand an erforderlichen Bauelementen wesentlich geringer, andererseits lassen sich auf Maschinen mit Parallelarbeitsweise größere Rechengeschwindigkeiten realisieren.

5. Grundbegriffe der Programmierung

5.1. Aufgabe der Programmierung

Rechenautomaten sollen Menschen von ständig wiederkehrenden formalen geistigen Arbeiten entlasten. Dazu ist es erforderlich, dem Automaten detaillierte Anweisungen zu geben, d.h. ihn zu programmieren. Dabei wird entsprechend einem vorliegenden Algorithmus dem Rechner mittels des Programms der Arbeitsablauf genau vorgeschrieben. Ein Programm ist demzufolge die Beschreibung eines Algorithmus in einer dem Rechenautomaten verständlichen Sprache, und unter Programmierung kann man im wesentlichen die Übersetzung aus einer Umgangs- oder Formelsprache in diese Sprache verstehen. Den Worten der Sprache entsprechen die vom Automaten ausführbaren Befehle. Die entscheidende menschliche Tätigkeit bei der Übernahme geistiger Routinearbeiten durch Rechenautomaten ist die Auffindung des Algorithmus dieser Arbeiten. Diese Tätigkeit kann von keiner Maschine übernommen werden.

5.2. Arbeitsweise bei der Programmierung

Bei der Bearbeitung eines Problems mit Hilfe eines Rechenautomaten kann man die folgenden Arbeitsgänge unterscheiden:

1. Formulierung und Beschreibung der Aufgabenstellung

   Das Problem muß mathematisch so formuliert werden, daß eine weitere rechnerische Behandlung möglich ist.

2. Auswahl eines Lösungsverfahrens

   Bei dieser Auswahl sollte bereits der zu benutzende Automatentyp berücksichtigt werden.

3. Erfassung des Berechnungsablaufes mit Hilfe eines Flußdiagramms (vgl. 1.3.)

4. Eigentliche Programmierung

(a) Aufstellung des Programms ohne feste Speicherplatzverteilung

(b) Organisation der Speicherverteilung

(c) Aufstellung des endgültigen Programms

(d) Herstellung des Eingabemediums

(e) Prüfen des Programms

Diese Phase der Programmherstellung hat besondere Bedeutung. Die, wie man im nächsten Abschnitt sehen kann, oftmals sehr komplizierte Struktur von Programmen verlangt sorgfältige Prüfverfahren, und das Auffinden von Programmfehlern ist meist sehr mühsam.

Die eigentliche Programmierung kann man sich erleichtern durch eine gründliche Vorbereitung und präzise Erfassung des Berechnungsablaufes. Von der Ausführung der in den Punkten 1. bis 3. genannten Arbeiten hängt die Anzahl der im Punkt 4. auftretenden Fehler wesentlich ab.

5.3. Struktur von Programmen

5.3.1. Geradeausprogramm und zyklisches Programm

Nach der Analyse des Problems und der Auswahl des Algorithmus wird dieser graphisch in Form eines Flußdiagramms dargestellt. In Abschnitt 1.3. ist die Leitlinienmethode erläutert, mit deren Hilfe diese Darstellung erfolgen kann. Ein Programm wird dann anhand des zugehörigen Flußdiagramms aufgestellt. Dieses bestimmt somit die Struktur und den Ablauf des Programms. Aus diesem Grunde bezeichnet man das Flußdiagramm auch als Programmablaufplan. Das Flußdiagramm bzw. der Programmablaufplan stellen also die Verbindung zwischen dem mathematischen Verfahren und dem Maschinenprogramm dar.

Aus den Abbildungen 23, 24 und 25 kann man entnehmen, daß die einfachsten Programmstrukturen durchgehende Linien ohne Verzweigungen sind. In diesem Fall spricht man von gestreckter Programmierung.

Komplizierter sind Programme, deren Ablaufpläne Verzweigungen (Maschen) besitzen (Abb. 24).

Wesentlich für die Struktur eines Programmes ist dabei, ob die Masche eine Rückführung (Schleife) innerhalb des Programmes darstellt, d.h., ob ein Programmteil mehrfach durchlaufen wird.

Programme dieser Art nennt man zyklisch, während Programme ohne Rückführung - auch wenn sie Maschen besitzen - als Geradeausprogramme bezeichnet werden (Abb. 23, 24, 25). Der Unterschied zwischen einem zyklischen und einem Geradeausprogramm besteht darin, daß bei dem Geradeausprogramm jeder Programmteil nur einmal abgearbeitet wird, während er bei zyklischen Programmen mehrfach durchlaufen werden kann.

Zyklische Programme sind für die Programmierung von großer Bedeutung. Sie sind wesentlich kürzer als Geradeausprogramme, da durch die mehrfache Verwendung eines Programmteiles die Anzahl der benötigten Befehle verringert wird. Anhand des Beispiels "Berechnung der Fakultät" soll das erläutert werden (vgl. Abschnitt 1.3.).

Würde man das Problem für n = 10 als Geradeausprogramm programmieren, so könnte der Programmablaufplan die Form haben, wie sie in Abb. 26 dargestellt ist.

Bei Verwendung eines zyklischen Programms werden die mehrfach auszuführenden Anweisungen

$$k := k + 1 \qquad \text{und} \qquad i := k \cdot i$$

nur einmal programmiert und man zählt, wie oft diese Anweisungen durchlaufen werden (Abb. 27).

Werden die beiden genannten Anweisungen das erste Mal durchgeführt, so wird gebildet

$$k := 0 + 1 = 1$$
$$i := 1 \cdot 1 = 1$$

Im zweiten Durchlauf ergibt sich

$$k := 1 + 1 = 2$$
$$i := 2 \cdot 1 = 2$$

Solange $k < 10$ ist, wird der Zyklus wiederholt. Erst bei $k = 10$, d.h. nachdem $i := 10 \cdot i$ gebildet worden ist, bricht die Rechnung ab.

Die zyklischen Programme lassen sich in zwei Gruppen einteilen:

A 1. Zyklen mit einer festen Anzahl von Durchläufen

Bei diesen Programmen steht die Anzahl der Durchläufe bereits zu Beginn der Rechnung fest (Abb. 78).

A 2. Zyklen mit einer variablen Anzahl von Durchläufen

Hier hängt die Zahl der Durchläufe von Größen ab, die erst während der Rechnung ermittelt werden (Abb. 8a).

Man kann die zyklischen Programme auch nach einem anderen Gesichtspunkt ordnen:

B 1. Induktionszyklen

Programme dieser Art verarbeiten in jedem Durchlauf nach den gleichen Anweisungen eine neue Gruppe von Parametern. Dazu sind in jedem Durchlauf systematische Änderungen der im Zyklus enthaltenen Befehle (d.h. deren Adresse) notwendig (vgl. 5.4.3.).

B 2. Iterationszyklen

Hier werden in jedem Durchlauf die gleichen Parameter nach einer bestimmten Rechenvorschrift verarbeitet.

Im allgemeinen haben Induktionszyklen eine feste und Iterationszyklen eine variable Anzahl von Durchläufen.

5.3.2. Unterprogramme

Häufig kommen in einem Programm an verschiedenen Stellen gleichartige Programmteile vor. So kann bei einem umfangreicheren Problem z.B. mehrmals die Berechnung einer Quadratwurzel auftreten. Man kann dann entweder an jeder Stelle, an der die Berechnung der Quadratwurzel benötigt wird, einen entsprechenden Programmteil einfügen oder man programmiert das Teilproblem nur einmal. Jedesmal, wenn eine Quadratwurzel zu berechnen ist, wird dieses Teilprogramm aufgerufen, abgearbeitet und danach in das übergeordnete Programm zurückgekehrt. Derartige Teilprogramme bezeichnet man als Unterprogramme.
Durch Unterprogramme wird Speicherraum eingespart und außerdem können häufig auftretende Teilprobleme als Unterprogramme programmiert werden, welche beliebig oft verwendbar sind. Diese Unterprogramme, die gewisse Standards erfüllen müssen, bezeichnet man als "Bibliothek[illegible]". (Im Anhang C ist eine Beschreibungssystematik für die Bibliotheksprogramme des U 8205 angegeben.) Zu jedem Rechner gehört im allgemeinen eine derartige Unterprogrammbibliothek, die Unterprogramme für mathematische und ökonomische Probleme, wie z.B. die Lösung von Gleichungssystemen, die Berechnung der elementaren Funktionen, die Multiplikation und Inversion von Matrizen usw., umfaßt.

Bei der Aufstellung und Verwendung eines Unterprogramms sind bestimmte Regeln zu beachten, die im folgenden erläutert werden sollen.

Für den Verkehr zwischen dem Unterprogramm und dem übergeordneten Programm, das im folgenden auch als Hauptprogramm bezeichnet werden soll, sind die sogenannten Parameter des Unterprogramms wesentlich. Das sind Größen, die vom Unterprogramm als Ausgangswerte der Rechnung benötigt oder als Ergebnis vom Unterprogramm an das Hauptprogramm übermittelt werden. Bei der Berechnung der Quadratwurzel z.B. ist x der zu verarbeitende Parameter und $\sqrt{x}$ der Ergebnisparameter. Dabei muß noch zwischen den Parametern, die die Adressen der Ausgangswerte bezeichnen und denen, die selbst Ausgangswerte sind, unterschieden werden. So muß bei der Berechnung des Skalarproduktes zweier Vektoren dem entsprechenden Unterprogramm mitgeteilt werden, wo die beiden Vektoren zu finden sind, und es muß die Dimension der Vektoren angegeben werden. Dem Hauptprogramm wird vom Unterprogramm als Parameter das Skalarprodukt übermittelt.

Ein weiteres Problem stellt die Organisation des Überganges zwischen Hauptprogramm und Unterprogramm dar. Der Übergang erfolgt durch einen Sprungbefehl, während die Rückkehr vom Unterprogramm zum Hauptprogramm mit Hilfe eines weiteren Parameters realisiert wird. Bei dem Einsprung in das Unterprogramm wird diesem die Rückkehradresse angegeben. Beim C 8205 gibt es, wie bei vielen Rechenautomaten, zur Bereitstellung des Rücksprungbefehles oder der Rücksprungadresse spezielle Befehle.

Ein übergeordnetes Programm muß einem Unterprogramm somit mitteilen

- welche Parameter durch das Unterprogramm zu verarbeiten sind und wohin die Ergebnisse zu speichern sind;
- an welche Stelle des übergeordneten Programms nach Beendigung des Unterprogramms zurückzukehren ist.

Bei den Unterprogrammen unterscheidet man zwischen offenen und geschlossenen Unterprogrammen.

Die offenen Unterprogramme sind unveränderlich auf festen Speicherplätzen untergebracht, während die geschlossenen Unterprogramme als Block an einer beliebigen Stelle gespeichert werden können.

5.4. Adressenänderungen

Im Abschnitt 5.3.1. wurde bei der Einteilung der Programme in Programme mit Induktions- und Iterationszyklen bereits von der Änderung der Adresse eines Befehls gesprochen.

Auch die Parameterversorgung von Unterprogrammen ist mit Adressenänderungen verbunden.

Diese Änderungen spielen bei der Programmierung eine wichtige Rolle, da sie einen wesentlichen Teil der Programmierungsarbeit ausmachen. Sie sollen im folgenden untersucht und klassifiziert werden.

5.4.1. Adressenänderungen 1. Art

Bei der Aufstellung eines Programms bekommt der erste Befehl im allgemeinen die Nummer Null zugeordnet und die Adressen der anderen Befehle, die sich auf das gleiche Programm beziehen, werden relativ zur Nummer des ersten Befehls angegeben.

Wenn dieses Programm ab einer Adresse gespeichert wird, die verschieden von Null ist - wie es in der Praxis meist der Fall sein wird - so müssen diese relativen Adressen geändert werden.

Hat man z.B. ein Programm von 0 anfangend numeriert und wird dieses Programm später ab der Zelle 100 gespeichert, so muß die Differenz 100 - 0 = 100 zu allen abzuändernden Adressen addiert werden.

Da diese Adressenänderung durch die Verschiebung des Programms im Speicher bedingt wird, bezeichnet

man sie auch als "Adressentranslation" oder als "Adressenänderung 1. Art".

Um eine Adressenänderung 1. Art durchführen zu können, müssen die zu verändernden Befehle besonders gekennzeichnet sein. Die Adressenänderung kann während der Eingabe der Programme oder auch erst unmittelbar vor Ausführung der Befehle vorgenommen werden. Dabei ist eine Änderung während der Eingabe - wie beim C 8205 - einfacher zu realisieren, da dann diese Kennzeichen nur auf dem Eingabemedium (z.B. Lochstreifen) angegeben werden, aber nicht mit im Speicher vermerkt sein müssen.

5.4.2. Adressenänderung 2. Art

In einem Unterprogramm können Adressen auftreten, die sich auf feste Speicherzellen oder das gleiche Unterprogramm beziehen sowie Adressen, die das übergeordnete Programm betreffen. Letztere sind beim Aufruf des Unterprogramms von verschiedenen Stellen des übergeordneten Programms unterschiedlich. Ihre Änderungen werden in der Regel vom übergeordneten Programm aus erledigt.

So muß z.B. bei der mehrmaligen Verwendung eines Unterprogramms in einem Hauptprogramm der Rücksprung aus dem Unterprogramm zu mehreren verschiedenen Adressen erfolgen, d.h. die Adresse des Rücksprungbefehles muß bei jedem Aufruf des Unterprogramms so abgeändert werden, daß nach Abarbeitung des Unterprogramms an die richtige Stelle des Hauptprogramms zurückgekehrt wird.

Wird das Unterprogramm "Matrizenmultiplikation" verwendet, so müssen dem Unterprogramm als Parameter die Anfangsadressen der Zahlenfelder für die beiden zu multiplizierenden Matrizen und die Ergebnismatrix sowie die Dimensionen der Felder vom Hauptprogramm übermittelt werden. In Abhängigkeit von diesen Parametern müssen dann die Adressen der entsprechenden Befehle im Unterprogramm durch die

tatsächlich benötigten Adressen ersetzt werden.

Diese mit der Parameterversorgung verbundenen und vom Aufruf des Unterprogramms abhängigen Veränderungen von Adressen bezeichnet man als Adressenänderungen 2. Art.

5.4.3. Adressenänderungen 3. Art

Die Adressenänderungen 3. Art sollen anhand eines Beispiels erläutert werden. Dazu wird das Programm "Berechnung des Skalarproduktes zweier Vektoren" betrachtet. Den Programmablaufplan zeigt Abb. 28.

In dem auftretenden Zyklus werden ein Zähler i bei jedem Durchlauf um eine Einheit erhöht, anschliessend die zugehörigen Komponenten a_i und b_i multipliziert und das Produkt zu der bereits berechneten Zwischensumme s addiert. Vor der Multiplikation der beiden Vektorkomponenten a_i und b_i müssen diese aus dem Speicher in das Rechenwerk überführt werden. Wenn angenommen wird, daß sowohl die Komponenten a_i, als auch b_i im Speicher hintereinander stehen, so müssen die Adressen der entsprechenden Befehle in jedem Durchlauf um eine Adresseneinheit erhöht werden. Damit wird in jedem Durchlauf ein neues Wertepaar a_i, b_i multipliziert.

Unter den Adressenänderungen 3. Art versteht man alle bei der Abwicklung induktiver Zyklen (vgl. 5.3.1.) notwendig werdenden Adressenänderungen.

5.5. Umsetzung von allgemeingültigen Algorithmen in die Maschinensprache

5.5.1. Manuelle Umsetzung

Das Programmieren von größeren Problemen ist mitunter äußerst mühsam und erfordert erhebliche Zeit. Fast jedes neue Programm enthält Fehler, deren Auffinden meist große Schwierigkeiten bereitet. Gründe dafür sind die elementare Struktur und die

damit verbundene Unhandlichkeit der Maschinensprache vieler Rechner.

Es ist deshalb wichtig, nach rationellen Arbeitsmethoden zu suchen. Ein Hilfsmittel ist die in Form von Unter- und Hauptprogrammen gespeicherte, geistige Arbeit anderer. In einer Zentralen Programmbibliothek werden allgemeininteressierende Programme von den Benutzern eines Maschinentyps zusammengetragen. Damit wird Doppelarbeit vermieden. In der Programmbibliothek für den C 8205 sind die im Anhang B aufgeführten Programme enthalten. Dieses Ziel erfordert jedoch eine einheitliche Gestaltung und Beschreibung der Programme. Im Anhang C wird deshalb die Beschreibungssystematik für Bibliotheksprogramme wiedergegeben.

5.5.2. Interpretationssysteme

Besonders bei Automaten, bei denen nur sehr elementare Befehle verdrahtet sind, wie z.B. beim C 8205, wäre es sehr unzweckmäßig, das Programm ausschließlich in Maschinenbefehle umzusetzen. Man führt deshalb Pseudo- oder Externbefehle ein. Einem derartigen Pseudobefehl entspricht dabei häufig nicht genau ein Maschinenbefehl, sondern ein Unterprogramm in der Maschinensprache.

Vor der Abarbeitung eines Programmes, das Pseudobefehle enthält, müssen diese jedoch in Maschinenbefehle umgewandelt werden. Das kann entweder so geschehen, daß vor der Abarbeitung eines Programms alle darin vorkommenden Pseudobefehle in Maschinenbefehle umgestellt werden, oder daß erst während der Abarbeitung des Programms ein auftretender Pseudobefehl in die Maschinensprache übersetzt und anschließend sofort ausgeführt wird.

Im ersten Fall könnte die Umstellung der Pseudo- in Maschinenbefehle z.B. bereits während der Eingabe des Programms vorgenommen werden. Erfolgt die Ausführung eines Pseudobefehls, wie im zweiten Fall,

unmittelbar nach seiner Übersetzung, so spricht man auch von einer interpretierenden Abarbeitung des Programms.

Alle dafür erforderlichen Programme sowie verschiedene Hilfs- und Kontrollprogramme sind in einem Programmsystem, dem sogenannten Interpretationssystem, zusammengefaßt. So gibt es z.B. beim C 8205 Interpretationssysteme zur Verarbeitung von Fest- und Gleitkommazahlen.

Ein solches Interpretationssystem stellt für den Benutzer einen neuen Rechenautomaten dar, der ein günstig zu handhabendes Befehlssystem besitzt. Der Vorteil gegenüber einer festen Verdrahtung besteht in der Austauschbarkeit der Systeme, so daß ein dem jeweiligen Problemkreis angepaßtes Befehlssystem verwendet werden kann.

5.5.3. Automatische Umsetzung maschinenabhängiger algorithmischer Eingangssprachen

Die bei der Programmierung eines Problems in der Maschinensprache auftretenden Schwierigkeiten waren der Grund, daß man sich seit langem bemüht, die Probleme in solchen Sprachen zu formulieren, die unabhängig von einem bestimmten Rechnertyp sind und die in diesen sogenannten problemorientierten Sprachen geschriebenen Programme dann von dem Rechenautomaten selbst in die Maschinensprache übersetzen zu lassen. So gibt es z.B. Sprachen für technisch-wissenschaftliche Probleme (FORTRAN, ALGOL) und für kaufmännische Verfahren (COBOL).

Besonderes Interesse von diesen Sprachen hat ALGOL gefunden. Diese Sprache dient zur algorithmischen Formulierung von Problemen (ALGOL = Algorithmic Language) und nimmt keine Rücksicht auf Maschineneigenschaften. Ein Programm, das in dieser Sprache geschrieben ist, ist wesentlich leichter zu lesen und abzuändern als ein Maschinenprogramm.

Für die Anwendung von problemorientierten Sprachen ist natürlich notwendig, daß Übersetzungsprogramme - sogenannte Compiler - vorhanden sind, so daß der Rechner aus dem Algolprogramm das zugehörige Maschinenprogramm erzeugen kann.

Bei kleineren Rechenautomaten, wie zum Beispiel beim C 8205, ergeben sich aus der Notwendigkeit der Übersetzungsprogramme, bei Verwendung von problemorientierten Programmsprachen, zwei Schwierigkeiten:

1. Die Übersetzungsprogramme benötigen einen relativ großen Speicherraum, der die Kapazität kleinerer Anlagen übersteigt.
2. Maschinell übersetzte Programme sind im allgemeinen nicht speicherplatz- und zeitgünstig.

Aus diesen Gründen erfordern hier verwendete problemorientierte Sprachen einschränkende Bedingungen.

Häufig verwandte Programme werden gewöhnlich manuell hergestellt und nur kurzfristig auftretende Aufgaben beschränkten Umfanges (z.B. in Entwicklungs- und Konstruktionsbüros) werden maschinell übersetzt.

S o f t - w a r e

I. Grundprogramme

1. Maschinenbedingte Programme

1.1. Programmierungshilfen

1.1.1. Bausteininterpretationssystem
1.1.2. Gleitkommainterpretationssystem
1.1.3. Festkommainterpretationssystem

1.2. Hilfsprogramme

1.2.1. Doppeln eines Lochstreifens im FS-Code
1.2.2. Ausschreiben eines Lochstreifens
1.2.3. Stanzprogramm für Festkommaausgabe zum Ausschreiben auf dem Schreibautomaten
1.2.4. Umwandlung Festkomma (vorn) in Gleitkommazahlen
1.2.5. Umwandlung von Adresseneinheiten in Gleitkommazahlen
1.2.6. Duplizieren im R 300-Code
1.2.7. Textcodierung
1.2.8. Textdruck

2. Programme der numerischen Mathematik

2.1. Funktionswertberechnung

2.1.1. Trigonometrische Funktionen
sin x, cos x, tan x, cot x
2.1.2. Inverse trigonometrische Funktionen
arc sin x, arc cos x, arc tan x, arc cot x
2.1.3. Hyperbolische Funktionen
sin h x, cos h x, tan h x, cot h x
2.1.4. Inverse hyperbolische Funktionen Areafunktionen
2.1.5. Natürlicher Logarithmus von x
2.1.6. Exponentialfunktionen e^x
2.1.7. Quadratwurzel
2.1.8. Wurzelfunktion

2.1.9. Hornerschema - einzeilig

2.1.10. Hornerschema - zweizeilig

2.2. Lösung algebraischer Gleichungen

2.2.1. Lösung einer quadratischen Gleichung

2.2.2. Lösung einer algebraischen Gleichung n-ten Grades nach Bairstow

2.3. Gleichungssysteme

2.3.1. Lösung eines linearen Gleichungssystems nach Gauß-Banachiewicz mit einfacher Pivotisierung

2.4. Matrizenrechnung

2.4.1. Matrizenaddition

2.4.2. Matrizenmultiplikation

2.4.3. Matrizeninversion

2.4.4. Transponierung einer Matrix

2.4.5. Matrix x Vektor

2.4.6. Determinantenberechnung

2.5. Vektorrechnung

2.5.1. Vektoraddition

2.5.2. Skalarprodukt

2.6. Differentialgleichungen

2.6.1. Differentialgleichung 1. Ordnung n. Runge-Kutta

2.6.2. Differentialgleichung 2. Ordnung n. Runge-Kutta

2.6.3. System von Differentialgleichungen 1. Ordnung nach Runge-Kutta

2.7. Numerische Integration

2.7.1. Trapezmethode nach Romberg

2.8. Statistik

2.8.1. Berechnung von Mittelwert, Streuung und Variationskoeffizienten

2.8.2. Berechnung des einfachen Korrelationskoeffizienten

2.8.3. Berechnung der Regressionsgeraden

2.8.4. Bestimmung des Minimums und des Maximums einer Zahlenfolge im Festkomma

2.9. Ausgleichsrechnung

2.9.1. Bestimmung der Ausgleichsparabel n-ten Grades

2.10. Sonstige mathematische Hilfsprogramme

2.10.1. Abrunden von Gleitkommazahlen

2.10.2. Runden von Gleitkommazahlen

2.10.3. Umwandlung karthesischer- in Polarkoordinaten

II. Verkaufsprogramme

1. Hilfsprogramme

1.1. Ganzer Teil einer Gleitkommazahl

1.2. Gebrochener Teil einer Gleitkommazahl

1.3. Umwandlung von Festkommazahlen (Komma vorn oder hinten) in Gleitkommazahlen

1.4. Umwandlung von Gleitkommazahlen in Adressen-einheiten

1.5. Stanzen einer Gleitkommazahl im SER-Code

2. Mathematische Programme

2.1. Gamma-Funktion

2.2. Besselfunktion $J_0(x)$, $J_1(x)$

2.3. Normalverteilung

2.4. Poissonverteilung

2.5. χ^2-Verteilung

2.6. Mehrfache lineare Regression

2.7. Harmonische Analyse für analytisch gegebene Funktionen

2.8. Harmonische Analyse für punktweise gegebene Funktionen

2.9. Inversion einer symmetrischen Matrix

2.10. Inversion einer Dreiecksmatrix

2.11. Lösung eines linearen Gleichungssystems mit dorgal-ber Pivotisierung und Test auf lin. Abhängigkeit

2.12. Jacobi-Verfahren

2.13. Mises-Iteration

2.14. System von Differentialgleichungen 1. Ordnung

2.15. χ^2-Test

2.16. Berechnung des einfachen Korrelationskoeffi-zienten mit Hilfe der Häufigkeiten

2.17. Holzer-Verfahren

3. Programme der Ökonometrie

3.1. PERT - Netzwerktechnik

3.2. Simplex-Verfahren

Anhang C

Programmbeschreibungen für Cellatron-Kleinrechenautomaten der Reihe D 4a

S y s t e m a t i k

Die Systematik soll einen raschen Programmaustausch zwischen Benutzern gleichartiger Automaten gewährleisten und die Anwendung von Programmen der zentralen Programmbibliothek beim VEB Kombinat Robotron erleichtern.

Programme, die der zentralen Programmbibliothek angeboten werden, sollen auf Grund des angewandten Vervielfältigungsverfahrens möglichst auf weißem Papier hergestellt werden. Bei Texten ist ein schwarzes Farbband in der Schreibmaschine zu verwenden. Tuschezeichnungen können auf Transparentpapier angefertigt werden.

Im einzelnen soll die Programmbeschreibung folgende Angaben enthalten:

1. Aufgabenstellung

   Auf einem Deckblatt wird eine kurze Darlegung des Problems gegeben. Es sollen möglichst alle Angaben vorhanden sein, die Außenstehenden die Entscheidung ermöglichen, ob sie das Programm für ihre Aufgaben verwenden können.

   Sind umfangreichere Darstellungen nötig, wird auf dem Deckblatt eine Kurzfassung gegeben, die auf den folgenden (formlosen) Blättern näher erläutert wird.

2. Algorithmus

3. Literaturhinweise

4. Benötigte Eingabedaten

5. Ergebnisse

6. Einsatzbereich

7. Rechenzeit

8. Zahlenbereich

9. Genauigkeit

   Bei der Fehleruntersuchung sind nach Möglichkeit Verfahrensfehler und Rechenfehler zu beachten. Beim Testen des Programmes sind ungünstige Fälle zu berücksichtigen.

10. Programmlänge

11. Benutzte Unterprogramme

12. Speicherplatzbelegung

    Programm (auf Formularen)
    Arbeitsspeicherplätze
    Versorgungsspeicherplätze

13. Speicherplatzbedarf

14. Anschlußbedingungen (bei Verwendung als Standard- oder Unterprogramm)

15. Programmablaufplan

    Für die Gestaltung des Programmablaufplanes ist die TGL 22 451 "Informationsverarbeitung - Datenfluß- und Programmablaufpläne - Sinnbilder" maßgebend.

    Für Ergibtanweisungen ist das Symbol ":=" zu verwenden, das Resultat steht auf der linken Seite.

    In den Programmablaufplänen sollen grundsätzlich keine maschinenbedingten Darstellungen verwendet werden. Neue Symbole sind grundsätzlich mit dem VEB Kombinat Robotron (Zentrale Programmbibliothek) zu vereinbaren.

Vereinbarung

Verschiebungen werden in der Weise a(b, c) dargestellt mit

a = Verschiebesymbol
    Vr = Rechtsverschiebung
    Vl = Linksverschiebung
    Vzl = zyklische Linksverschiebung

b = Zahl der Verschiebungen (Konstante, Parameter oder Variable)

c = zu verschiebender Wert

Zur näheren Erläuterung des Programmablaufplanes können Details gesondert dargestellt werden. In den Detaildarstellungen sind folgende Angaben möglich:

a) Angaben, die im Hauptprogrammablaufplan zulässig sind

b) Maschinenbefehle

c) besonders vereinbarte Zeichen

Vereinbarung

Soll in der Detaildarstellung die Abarbeitung eines Befehles, der erst durch frühere Operationen gebildet wurde, gezeigt werden, ist folgende Darstellung zulässig:

a := b Op c

a = Resultat der Operation

b = Befehl

c = zu verarbeitender Wert

16. Bemerkungen

16. Bemerkungen

Unter diesem Punkt sind unter anderem die Konstanten, die aus dem Interpretationssystem verwendet werden, anzugeben.

17. Testbeispiel

18. Bedienungsanleitung

Die Bedienungsanleitung soll alle für die Bedienungskraft erforderlichen Hinweise enthalten. Sie soll von der übrigen Programmbeschreibung abtrennbar sein.

Formale Beschreibung von Sprachen, Normalform von Backus

Um die Beschreibung einer Sprache abzufassen, ist diese Sprache selbst ungeeignet; denn es entsteht dabei die Schwierigkeit, daß man Zeichen, Wörter und grammatische Regeln benutzen muß, die erst noch definiert werden müssen. In ähnlicher Lage befindet man sich bei der Beschreibung von Programmierungssprachen. Da diese im allgemeinen auch Zeichen und Wörter der Umgangssprache enthalten, können bei einer umgangssprachlichen Beschreibung ebenfalls Mehrdeutigkeiten und Irrtümer entstehen.

Um solche Schwierigkeiten zu vermeiden, werden zur Beschreibung von Sprachen sogenannte Metasprachen, die unabhängig von der zu beschreibenden Sprache sind, benutzt. Eine zur Beschreibung von Programmierungssprachen gut geeignete Metasprache ist die Backussche Normalform.

1. Backussche Normalform

In der Backusschen Normalform kommen vor:

Definitionszeichen ::=

Oder-Zeichen |

Klammern < >

Namen und Symbole

Definitionszeichen, Oder-Zeichen und Klammern dürfen in der zu beschreibenden Sprache nicht vorkommen. Namen und Begriffe der zu beschreibenden Sprache werden durch Klammersetzung, z.B. <Name>, aus dieser herausgenommen; sie können in dieser Form als Beschreibungsmittel benutzt werden. (Zur Vereinfachung mögen hier die Namen aus Zeichen der Umgangssprache bestehen.) Symbole haben nicht die Bedeutung, die ihnen in der zu beschreibenden Sprache zukommt, sondern geben nur an, daß an dieser Stelle das hingeschriebene Symbol steht.

Zur Beschreibung der Backusschen Normalform werden noch die beiden Zeichen

<- mit der Bedeutung ::=
| mit der Bedeutung |

benutzt.

Die Syntax der Backusschen Normalform lautet damit:

⟨Backussche Normalform⟩ <- ⟨Subjekt⟩::=⟨Prädikat⟩|⟨Subjekt⟩::=

d.h., die Backussche Normalform wird definiert durch ein Subjekt, gefolgt von einem Definitionszeichen und einem Prädikat, oder durch ein Subjekt, gefolgt von einem Definitionszeichen;

⟨Subjekt⟩ <- ⟨Name⟩
⟨Prädikat⟩ <- ⟨Name⟩ | ⟨Symbol⟩ | ⟨Prädikat⟩ ⟨Prädikat⟩ |
⟨Prädikat⟩ | ⟨Prädikat⟩

d.h., ein Prädikat wird definiert durch einen Namen oder durch ein Symbol oder durch ein Prädikat, gefolgt von einem Prädikat, oder durch ein Prädikat, gefolgt von einem Oder-Zeichen und einem Prädikat.

2. Beispiele

Die folgenden Beispiele geben keine für die praktische Arbeit ausreichende Beschreibung, sondern sollen nur zur Illustration der Backusschen Normalform dienen (Heft 2, Anleitung zur Programmierung in der Maschinensprache).

2.1. Maschinenwort des C 8205

⟨Grundsymbole⟩ ::= 0|1
⟨Maschinenwort⟩ ::= ⟨Zahlwort⟩|⟨Befehlswort⟩
⟨Zahlwort⟩::= ⟨Triade⟩ ⟨Triade⟩ ⟨Triade⟩ ⟨Triade⟩
⟨Triade⟩ ⟨Triade⟩ ⟨Triade⟩ ⟨Triade⟩
⟨Triade⟩ ⟨Triade⟩ ⟨Triade⟩
⟨Triade⟩ ::= ⟨Bit⟩ ⟨Bit⟩ ⟨Bit⟩
⟨Bit⟩ ::= 0|1

⟨Befehlswort⟩ ::= ⟨vordere Bits⟩ ⟨Befehl⟩

⟨Befehl⟩ ::= ⟨Adresse⟩ ⟨Operationsteil⟩

⟨Adresse⟩ ::= ⟨Triade⟩⟨Triade⟩⟨Triade⟩⟨Triade⟩

⟨Operationsteil⟩ ::= ⟨Befehlstyp⟩⟨Operationsart⟩
⟨Operationszusatz⟩

⟨Befehlstyp⟩ ::= ⟨Triade⟩

⟨Operationsart⟩ ::= ⟨Triade⟩

⟨Operationszusatz⟩ ::= ⟨Resultatbehandlung⟩⟨Löschbit⟩

⟨Resultatbehandlg⟩ ::= ⟨Bit⟩⟨Bit⟩

⟨Löschbit⟩ ::= ⟨Bit⟩

2.2. Eingabeworte des kleinen Eingabeprogramms (KEP)

⟨Grundsymbole⟩ ::= 0|1|ZR|.|:

⟨KEP-Wort⟩ ::= ⟨Leitbefehl⟩|⟨Speicherwort⟩|⟨falsches Wort⟩

⟨Leitbefehl⟩ ::= ⟨Bitfolge⟩ :

⟨Speicherwort⟩ ::= ⟨Bitfolge⟩ .

⟨falsches Wort⟩ ::= ⟨Bitfolge⟩ ZR

⟨Bitfolge⟩ ::= ⟨leere Folge⟩|⟨Bitfolge⟩ ⟨Bit⟩

⟨leere Folge⟩ ::=

⟨Bit⟩ ::= 0|1

Literaturverzeichnis

Götzke, Horst: Programmgesteuerte Rechenautomaten
3. Auflage 1967
Leipzig: VEB Fachbuchverlag

Bachmann, K-H.: Programmierung für Digitalrechner
Berlin: VEB Deutscher Verlag der Wissenschaften (1962)

Bär, Dieter: Einführung in die Schaltalgebra
Reihe Automatisierungstechnik, Band 25
Berlin: VEB Verlag Technik

Güntsch, Fr. Rudolf: Einführung in die Programmierung digitaler Rechenautomaten
2. Auflage 1963
Berlin: Walter de Gruyter & Co.

Schubert, Günter: Digitale Kleinrechner
Reihe Automatisierungstechnik, Band 5
Berlin: VEB Verlag Technik

Paulin, Gerhard: Kleines Lexikon der Rechentechnik und Datenverarbeitung
Reihe Automatisierungstechnik, Band 52
Berlin: VEB Verlag Technik

Materialien der Forschungsstelle für Truppenführung der NVA:

- Programmierungsanleitung für den Digitalrechner D 4a 1965